Test Case Migration: A Reference Process Model and its Instantiation in an Industrial Context

Ivan Jovanovikj, Enes Yigitbas, Stefan Sauer [1]

Abstract: In the context of software migration, existing test cases represent important assets which can be reused for migration validation. Doing so, one can save time as well as costs by avoiding design of new test cases. However, migration of test cases is a quite challenging task due to the size of the test case set and often missing conformity in the structure of the test cases. Moreover, the test cases are implemented in the same or a compatible technology as the software system they are testing, so they have to follow somehow the migration of the system. To overcome this complex task, we propose a reference process model which supports systematic test case migration during software migration projects. Our reference process model for test case migration encompasses the phases reverse engineering, restructuring, and forward engineering. An instantiation of our reference process model is presented based on an industrial case study.

Keywords: software migration; reengineering; model-driven testing

1 Introduction

Software migration is a well-established method for transferring software systems into new environments without changing their functionality. For instance, legacy software systems which are technologically obsolete, but still valuable for ongoing business, can be migrated into a new environment [Bi99].

In this context, software testing plays an important role as it verifies whether the main requirement, the preservation of the functionality, is fulfilled. Reusing existing test cases can be beneficial not just from economical perspective, as the expensive and time consuming test design is avoided [Sn99], but also from practical perspective: the existing test cases contain valuable information about the functionality of the original system.

However, a direct reuse of existing test case is not always possible in real world migration projects, due to system changes as well as the differences in the source and the target environments. As test cases are coupled with the system they are testing, the system changes need to be detected, understood and then reflected on the test cases to facilitate reuse. This is a quite challenging task in various migration scenarios, because of the size of the test

[1] Paderborn University, Software Innovation Lab, Zukunftsmeile 1, 33102 Paderborn, Germany, {ivan.jovanovikj|enes.yigitbas|stefan.sauer}@sicp.upb.de

case set which sometimes is measured in tens of thousands of test cases and the missing conformity in the test case structure [JGY16]. Hence, first of all, a new migration method should be able to deal with the structural complexity of the test cases. Then, it should also provide an easy way to restructure the test cases and reflect the relevant system changes. Last but not least, the migrated test cases should be appropriate for the target environment, i.e., the test cases have to be re-designed for the target environment [Gr16].

The Model-Driven Engineering (MDE) software development methodology has been established to deal with the increasing complexity of today's software systems by focusing on creating and exploiting domain models. The Model-Driven Architecture (MDA) initiative, proposed by Object Management Group (OMG), puts the MDE idea in action, and clearly separates the business logic from implementation by defining software models at different levels of abstraction: The computation independent model (CIM) layer, platform-independent model (PIM) layer and platform-specific model layer (PSM).

Software migration approaches that follow the MDA principles are known as model-driven software migration (MDSM) approaches [Fu12]. Generally speaking, software migration can be seen as a transformation of the system which is done by enacting a software transformation method, which is an instance of the well-known horseshoe model [KWC98]. Therefrom, software migration is some kind of software reengineering, which according to [CC90] is defined as "examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form". In general, software reengineering consists of three consecutive phases: reverse engineering, restructuring, and forward engineering. Software testing which relies on the MDA principles is known as model-driven testing [HL03, EGL06]. Similarly to software migration, test case migration can be seen as a transformation of test cases implemented for a specific source technology to test cases implemented for a specific target technology.

Following the idea of model-driven software migration, we present a novel reference process model which supports systematic test case migration during software migration projects. Our proposed reference process model encompasses the phases *Reverse Engineering*, *Restructuring*, and *Forward Engineering* (Figure 1). An instantiation of our reference process model is presented based on an industrial case study which illustrates the applicability of our reference process.

The structure of the rest of the paper is as follows: In the next section 2, we present our reference process model for test case migration. The applicability of our reference process model is shown in section 3 based on an industrial case study. In section 4, we briefly discuss the related work and at the end, section 5 concludes the work and gives an outlook on future work.

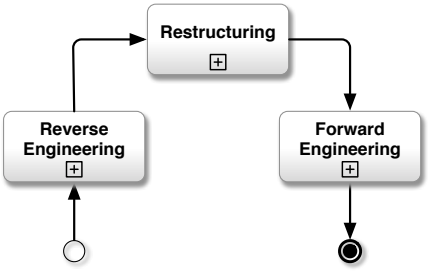

Fig. 1: General Phases of the Test Case Migration Process

2 Reference Process Model for Test Case Migration

In this section, we present the reference process for test case reengineering as enabler for model-driven test case migration. In general, as shown in Figure 1, our reference process comprises the main reengineering phases: *Reverse engineering*, *Restructuring*, and *Forward Engineering* [CC90]. Each phase consists of a set of activities and corresponding artifacts in terms of textual artifacts and models on different levels of abstraction. Here, we introduce the reference process represented as a BPMN model (Figure 2) and in the next section we come up with a concrete instantiation.

2.1 Artefacts

Artefacts are constituting parts of each transformation process. In Figure 2, the artefacts are represented as data objects. An artefact can either be a textual artefact, e.g., a test code, or a model. Regarding to the abstraction level, each artefact belongs to one of the three layers: *System Layer*, *Platform-Specific Layer*, and *Platform-Independent Layer*.

On the *System Layer*, the textual artifacts representing test code and models of the test code (*Model of the Test Code (original)* and *Model of the Test Code (migrated)*) are placed. *Test Code (original/migrated)* can be either the test cases, implemented in a specific testing framework, e.g. jUnit[2] or MSUnit[3], test configuration scripts or a manually implemented additional test framework. If the test cases depend on these artifacts, they also have to be considered and eventually migrated along with the test cases. *Model of the Test Code (original/migrated)* is an AST (Abstract Syntax Tree) representation of the test code depending on the particular language of the original or the target environment.

On *Platform-Specific Layer*, technology-specific test concepts are used to represent the test cases for both the source and the target environment. The *Model of Executable Tests (Original/Target)* is considered as platform-specific as it represents the executable test cases

[2] jUnit, http://junit.org/junit4/
[3] MSUnit, https://msdn.microsoft.com/en-us/library/hh598960.aspx

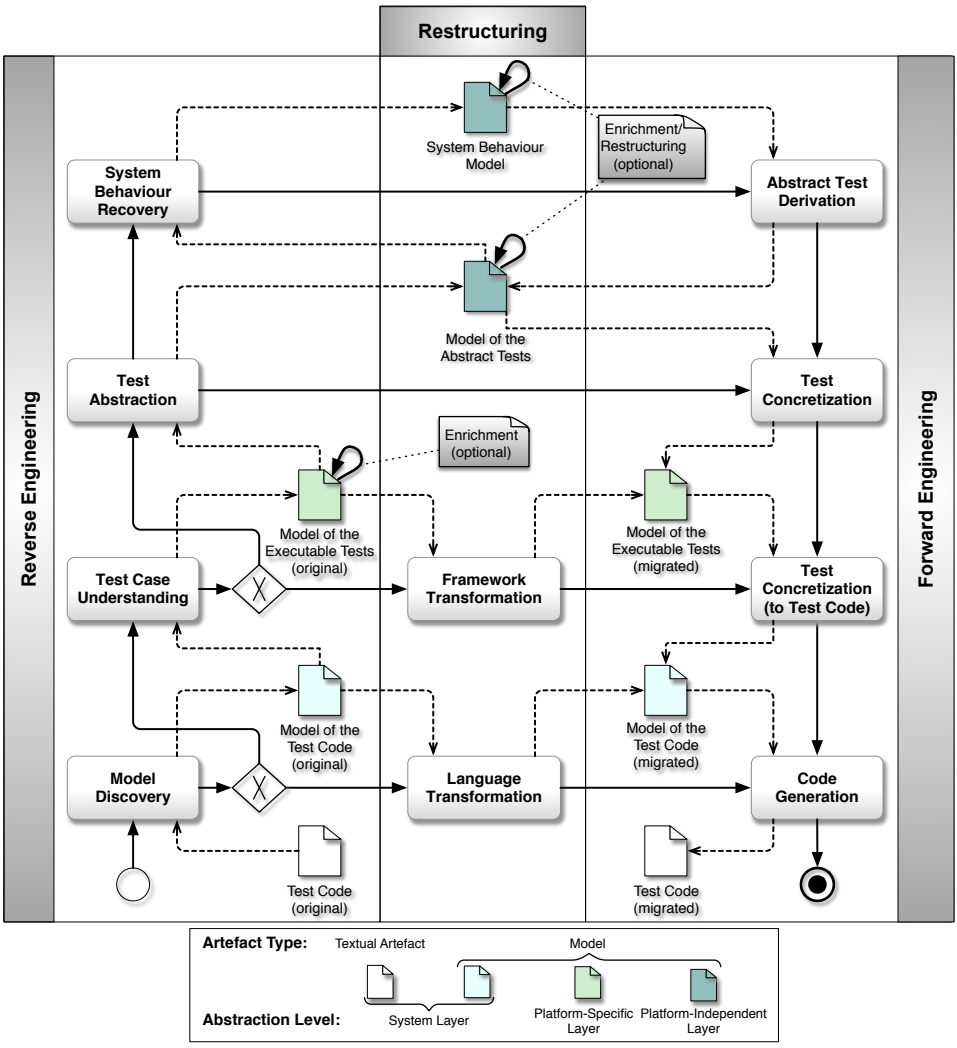

Fig. 2: Reference Process Model for Test Case Migration

by using testing concepts which are specific for a particular testing framework, i.e., by using meta-models of jUnit or MSUnit to model the tests.

On the *Platform-Independent Layer*, the models are independent of any particular testing framework or testing technology. Here, two types of models can be distinguished, the *Model of the Abstract Tests* and the *System Behavior Model*. The *Model of the Abstract Tests* is considered to be platform-independent as it is independent of any concrete testing technology. For the modeling of abstract test cases, standardized languages like UML

Testing Profile (UTP) or Test Description Language (TDL) are used. The *System Behavior Model* comes at the highest level of abstraction and it is a compact representation of the expected system's behavior usually represented as UML activity or sequence diagrams.

2.2 Activities

Activities in the test case reengineering process model produce or consume appropriate artifacts. These activities can be distinguished by the reengineering phase they are belonging to. For this purpose, we rely on the classification defined by [CC90], who define the reengineering as a process constituted of three phases: *Reverse Engineering*, *Restructuring*, and *Forward Engineering*. In Figure 2, all activities are represented as BPMN tasks.

Reverse Engineering is according to [CC90] the "process of analyzing a subject system to create representations of the system in another form or on a higher level of abstraction". Similarly, seen from software testing perspective, we define reverse engineering as a process of analyzing test cases to create representations of the test cases in another form or on a higher level of abstraction, e.g., by using test models. In general, reverse engineering can be seen as a combination of *Model Discovery* and *Model Understanding* [Br10]. *Model Discovery* is an automatic text-to-model transformation activity which relies on parsers to perform syntactical analysis. As a result, a model in terms of AST of the test case source code is produced. *Model Understanding* is a model-to-model transformation activity, or a chain of such activities, which take the initial model and apply semantic mappings in order to generate a derived model of a higher level of abstraction. In our reengineering process, *Model Understanding* comprises three sub-activities. Firstly, *Test Case Understanding* takes the initial models and explores them by navigating through their structure to identify test relevant concepts like test suite, test case or assertion. Then, in a model-to-model transformation step, a test model of executable test cases is obtained. This model is specific to a particular testing technology (e.g., jUnit, MSUnit). Next, by applying the *Test Abstraction* activity, a model-to-model transformation as well, a model of the abstract test cases is obtained. This model is a platform-independent representation of the test cases, thus independent of any particular testing technology. As modeling languages, usually UML Testing Profile (U2TP) or Test Description Language (TDL) are used. Finally, the *System Behavior Recovery* activity is applied in order to obtain the highest possible level of abstraction defined by our reengineering process, the *System Behavior Model*. It is a compact representation of the expected behavior of the system. For all of the previously mentioned model-to-model transformations, transformation languages like QVT[4] or Java Development Tools (JDT)[5] can be used.

Forward Engineering is "the traditional process of moving from high-level abstractions and logical, implementation-independent designs to the physical implementation of a

[4] QVT - http://www.omg.org/spec/QVT/1.1/
[5] JDT - https://www.eclipse.org/jdt/

system-[CC90]. In the testing domain, we define this as a process of moving of high-level test abstractions and logical implementation-independent design to the physical implementation of the test cases. The *Forward Engineering* phase can be seen as Model-Driven Testing [HL03, EGL06] as the test models are used as input for a chain of automated transformations, which at the end provides the test code. The *Abstract Tests Derivation* activity gets as input the *System Behavior Model* and generates the *Model of Abstract Test Cases*. Next, by applying *Test Concretization*, a *Model of Executable Test Cases* is obtained. The *Test Code Generation* activity, takes this platform-specific model and in a model-to-text transformation, test code of the migrated test cases is obtained.

Restructuring, according to [CC90], is "the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics)". In our case, we define test restructuring as the transformation from one representation to another at the same relative abstraction level, while preserving the ßemanticsöf the tests. We use "semantics" to denote the functionality of the system being checked by the tests. This activity can be applied on the *System Behavior Model* as well as on the *Model of Abstract Test Cases*. In Figure 2, we indicate *Restructuring* as "loop" symbols on the artifacts they are applicable on. The particular actions during *Restructuring* are defined by the target testing environment. But, *Restructuring* may also be influenced by the changes that happen in the system migration. Those system changes that are relevant for the tests have to be considered and covered by the *Restructuring* activity. *Language Transformation* is a direct mapping between the *Models of the Test Code*. More precisely, it is actually a mapping between the programming languages of the source and target testing environment. *Framework Transformation* is performed on a higher abstraction level and defines a direct mapping between two testing frameworks, i.e., a mapping between the test concepts inside the original an the target testing framework. In *Enrichment*, one can insert an additional information to the tests by using annotations. This activity is applicable to various models, e.g., *Model of Executable Tests*, *Model of Abstract Tests* or *System Behavior Model*.

The XOR-Gateways shown in Figure 2, illustrate the flexibility of the reengineering process and its applicability in different migration contexts. For example, in a concrete migration scenario, as shown in the next section, one can decide to perform *Framework Transformation*, and not *Language Transformation*. This implies that after *Model Discovery*, on the first decision node, a *Test Case Understanding* has to be chosen.

3 Case Study: An Instantiation of the Reference Process Model in an Industrial Context

Our reference reengineering process was applied in an industrial project which dealt with migration of parts of the well-known Eclipse Modeling Framework (EMF)[6] along with the

[6] Eclipse Modeling Framework, https://www.eclipse.org/modeling/emf/

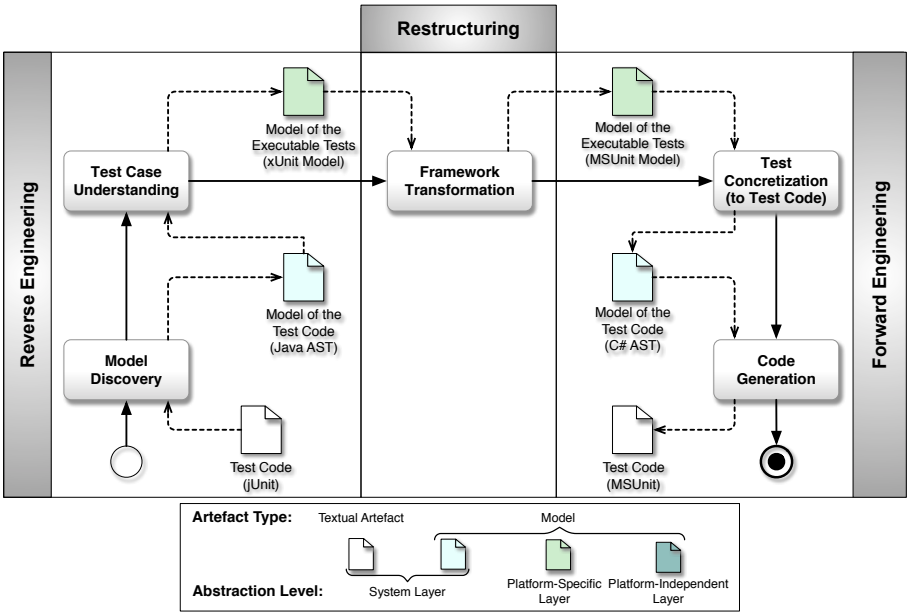

Fig. 3: Instantiation of the Reference Process Model for Test Case Migration

Object Constraint Language (OCL)[7] from Java to C#. The main change performed by the system migration, besides switching from Java to C#, was to change from a Just-In-Time (JIT) compilation in the old system to an Ahead-Of-Time (AOT) compilation of OCL constraints in the migrated system. As OCL is a well-tested framework, the main goal was to reuse the existing OCL test cases to validate the migrated OCL functionality in the target environment. Consequently, in total 13 different test suites had to be migrated, each of them addressing different functional aspects of OCL.

According to this specific context information, we used our test case reengineering process model shown in Figure 2 to instantiate a suitable reengineering method. At the end, the migration method shown in Figure 3 was obtained. The *Reverse Engineering* phase starts with *Model Discovery*, which takes as input jUnit test cases (*Test Code (jUnit)*) and produces an AST (*Model of the Test Code (Java AST)*). Then, *Test Case Understanding* takes the AST as input, and identifies and extracts testing relevant concepts, resulting in a *Model of the Executable Test Cases (xUnit)* which conforms to the xUnit meta-model [Me07]. In *Framework Transformation* a transformation based on mapping between xUnit and MSUnit is performed. Here, also the transformation of OCL expressions from Java to C# was performed. As this transformation was already implemented in the system migration, it was reused. Then, by applying *Test Concretization (to Test Code)* and *Code Generation* consecutively, the test code (*Test Code (MSUnit)*) for the target environment was obtained.

[7] Object Constraint Language, http://www.omg.org/spec/OCL/2.4/

To support the migration process, we developed two Eclipse plug-ins: *TestCase2TestModel* and *TestModel2TestCase*. The *TestCase2TestModel* plug-in supports the reverse engineering activities by using JDT for parsing and model-to-model transformation. The forward engineering and restructuring activities, are supported by the *TestModel2TestCase* plug-in which is a test code generator written using Xtend, resulting at the end in MSUnit test code.

In total, 92% of the 4000 existing jUnit test cases were migrated. It is an open question if the migration of the test cases can further be automatized. 8% of the OCL test cases were not migrated automatically, because these are regression tests that have an irregular structure which complicates an automation.

4 Related Work

Focussing on the topic of model-driven test case migration, different research areas like model-driven testing, test case reengineering, test mining and software migration have to be analyzed.

Previous research in the area of model-driven testing already presented some proved methods that address the automation of the test case generation in a quite efficient way. In the work presented in [HL03], the authors aim to benefit from the separation of PIMs and PSMs in the generation and execution of tests by redefining the classical model-based tasks. Dai [Da04] proposes a model-driven testing methodology which takes as input UML system models and transforms them to test-specific models as instances of the UML Testing Profile. Javed et al. [JSW07] propose a generic method for model-driven testing that relies on an xUnit platform independent model. Lamancha et al. [LA09] propose also a model-driven testing method that relies on UML Testing Profile. Moreover, they present concrete transformations using the QVT transformation language. All of this methods are related to our work as they address the forward engineering phase of our reference process model.

Regarding the reverse engineering side, we have identified also some existing work in the area known as test case reengineering. Hungar et al. [Hu03] extract models out of test cases by means of automata learning. In [Jä09], test models are synthesized from test cases by threating all test cases as a linear model and merging the corresponding states. The work of Werner et al. [WG11] goes in the similar direction and constructs trace graphs out of test cases to represent the system behavior.

The importance of reusing test cases can be observed in various previous software migration projects where effort has been made to enable test case reuse. In our previous work [Jo16], we have already presented the basic idea about reenginneering of legacy test cases. In the SOAMIG migration project [Zi11] for example, existing test cases are used for the analysis of the legacy system. MoDisco [Br10] is a generic and extensible framework devoted to Model-Driven Reverse Engineering. However, migration of test cases is still not addressed by this framework. The Artist [MA14] project proposes model-based modernization, by

employing MDE techniques to automate the reverse engineering and forward engineering phases. From our point of view, it is the most interesting project as they also advocate migration of the test cases in a model-driven way, i.e., in a similar way the system has been migrated, thus reusing, above all, the developed tools. Our work differs in that way, that we propose a systematic test case migration reference process model seen from software testing perspective.

5 Conclusion and Future Work

In this paper, we present a novel reference process model for test case migration during software migration projects. Our approach supports model-driven test case migration through a systematic description of relevant reengineering activities represented in BPMN. It comprises the reengineering phases *Reverse Engineering*, *Restructuring*, and *Forward Engineering* as well as artefacts on different levels of abstraction specific for software testing. The proposed reference process model is a generic model with high flexibility and can be used in different software migration scenarios to support the model-driven migration of test cases. This is shown by an instantiation of our reference process model based on an industrial case study where thousand of test cases were migrated from jUnit to MSUnit, most of them completely automatic. In future work, we intend to further extend our reference process model based on the idea of situational method engineering. This way, we plan to systematically cover different test case transformation methods suitable for different migration contexts.

References

[Bi99] Bisbal, J.; Lawless, D.; Bing Wu; Grimson, J.: Legacy Information Systems: Issues and Directions. IEEE Software, 16(5):103–111, 1999.

[Br10] Bruneliere, Hugo; Cabot, Jordi; Jouault, Frédéric; Madiot, Frédéric: MoDisco. In: Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10. ACM Press, New York, New York, USA, p. 173, 2010.

[CC90] Chikofsky, Elliot J; Cross, James H: Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software, 7(1):13–17, 1990.

[Da04] Dai, Zhen Ru: Model-Driven Testing with UML 2.0. Computing Laboratory, University of Kent, 2004.

[EGL06] Engels, Gregor; Güldali, Baris; Lohmann, Marc: Towards Model-Driven Unit Testing. In: Models in Software Engineering, pp. 182–192. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[Fu12] Fuhr, Andreas; Winter, Andreas; Erdmenger, Uwe; Horn, Tassilo; Kaiser, Uwe; Riediger, Volker; Teppe, Werner: Model-Driven Software Migration - Process Model, Tool Support and Application. In: Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments, pp. 153–184. 2012.

[Gr16] Grieger, Marvin: Model-Driven software modernization: concept-based engineering of situation-specific methods. PhD thesis, University of Paderborn, Germany, 2016.

[HL03] Heckel, Reiko; Lohmann, Marc: Towards model-driven testing. In: Electronic Notes in Theoretical Computer Science. volume 82. Elsevier, pp. 37–47, 9 2003.

[Hu03] Hungar, Hardi; Hungar, Hardi; Margaria, Tiziana; Steffen, Bernhard: Test-Based Model Generation for Legacy Systems. IEEE International Test Conference (ITC), Charlotte, NC, September 30 - October 2, 2:2003, 2003.

[Jä09] Jääskeläinen, Antti; Kervinen, Antti; Katara, Mika; Valmari, Antti; Virtanen, Heikki: Synthesizing Test Models from Test Cases. In: Hardware and Software: Verification and Testing: 4th International Haifa Verification Conference, pp. 179–193. 2009.

[JGY16] Jovanovikj, Ivan; Grieger, Marvin; Yigitbas, Enes: Towards a Model-Driven Method for Reusing Test Cases in Software Migration Projects. Softwaretechnik-Trends, Proceedings of the 18th Workshop Software-Reengineering & Evolution (WSRE) & 7th Workshop Design for Future (DFF), 32(2):65–66, 2016.

[Jo16] Jovanovikj, Ivan; Grieger, Marvin; Güldali, Baris; Teetz, Alexander: Reengineering of Legacy Test Cases: Problem Domain & Scenarios. Softwaretechnik-Trends, Proceedings of the 3rd Workshop Model-Based and Model-Driven Software Modernization (MMSM), 36(3):65–66, 2016.

[JSW07] Javed, A. Z.; Strooper, P. A.; Watson, G. N.: Automated Generation of Test Cases Using Model-Driven Architecture. In: Second International Workshop on Automation of Software Test (AST '07). IEEE, pp. 3–3, 5 2007.

[KWC98] Kazman, R.; Woods, S.G.; Carriere, S.J.: Requirements for integrating software architecture and reengineering models: CORUM II. In: Proceedings Fifth Working Conference on Reverse Engineering. IEEE Comput. Soc, pp. 154–163, 1998.

[LA09] Lamancha, Beatriz Pérez; Al., Et: Automated model-based testing using the UML testing profile and QVT. In: Proceedings of the 6th International Workshop on Model-Driven Engineering, Verification and Validation. pp. 1–10, 2009.

[MA14] Menychtas, Andreas; Al., Et: Software modernization and cloudification using the ARTIST migration methodology and framework. Scalable Computing: Practice and Experience, 15(2):131–152, 7 2014.

[Me07] Meszaros, Gerard.: XUnit test patterns : refactoring test code. Addison-Wesley, 2007.

[Sn99] Sneed, H.M.: Risks involved in reengineering projects. In: Sixth Working Conference on Reverse Engineering. pp. 204–211, 1999.

[WG11] Werner, Edith; Grabowski, Jens: Model Reconstruction: Mining Test Cases. In: Third International Conference on Advances in System Testing and Validation Lifecycle. VALID 2011, 10 2011.

[Zi11] Zillmann, C.; Winter, A.; Herget, A.; Teppe, W.; Theurer, M.; Fuhr, A.; Horn, T.; Riediger, V.; Erdmenger, U.; Kaiser, U.; Uhlig, D.; Zimmermann, Y.: The SOAMIG Process Model in Industrial Applications. In: 2011 15th European Conference on Software Maintenance and Reengineering. IEEE, pp. 339–342, 3 2011.